1998-09

Using Local Optimality Criteria for Efficient Information Retrieval with Redundant Information Filters

Neil C. Rowe

Monterey, California: Naval Postgraduate School.

arXiv:cs/9809121v1 [cs.IR] 29 Sep 1998

Using Local Optimality Criteria for Efficient Information Retrieval with Redundant Information Filters

Neil C. Rowe | REFERENCE 1|

Code CS/Rp, Department of Computer Science

Naval Postgraduate School Monterey, CA USA 93943 rowe@cs.nps.navy.mil

Abstract

We consider information retrieval when the data, for instance multimedia, is computationally expensive to fetch. Our approach uses "information filters" to considerably narrow the universe of possibilities before retrieval. We are especially interested in redundant information filters that save time over more general but more costly filters. Efficient retrieval requires that decisions must be made about the necessity, order, and concurrent processing of proposed filters (an "execution plan"). We develop simple polynomial-time local criteria for optimal execution plans, and show that most forms of concurrency are suboptimal with information filters. Although the general problem of finding an optimal execution plan is likely exponential in the number of filters, we show experimentally that our local optimality criteria, used in a polynomial-time algorithm, nearly always find the global optimum with 15 filters or less, a sufficient number of filters for most applications. Our methods do not require special hardware and avoid the high processor idleness that is characteristic of massive-parallelism solutions to this problem. We apply our ideas to an important application, information retrieval of captioned data using natural-language understanding, a problem for which the natural-language processing can be the bottleneck if not implemented well.

| REFERENCE 1| This work was sponsored by DARPA as part of the I3 Project under AO 8939, and by the U. S. Naval Postgraduate School under funds provided by the Chief for Naval Operations. Discussions with Amr Zaky improved this paper. Classification: H.3.3 (Information Search and Retrieval), Search Processes. Additional terms: filters, optimization, queries, conjunction, boolean algebra, natural language. This paper appeared in *ACM Transactions on Information Systems, 14*, no. 2 (April 1996), 138-174.

Introduction

We address the problem of efficient information retrieval of data that matches boolean query expressions. For example, for a database of captioned pictures, we could ask:

> Find side views of burros or donkeys in the desert where their image covers more than 10% of the picture, or else burros at the zoo where their image covers more than 20% of the picture.

which could be stated in a boolean query language as:

> ("side view" and ("burro" or "donkey") and "desert" and ((relativesize ("burro" or "donkey")) > 0.1)) or ("burro" and "zoo" and ((relativesize "burro") > 0.2) and "side view")

Different ways of processing this query could result in markedly different execution times. For instance, should we do the last line first? And the relative sizes are not likely to be mentioned in the caption and will require some time-consuming image processing, which probably should be done last, but how can we be sure? And if we are likely to get many queries mentioning animals in the desert, it would improve efficiency to have a redundant signature table for all of them, but how often must such queries occur for the hash table to pay off? We need quantitative criteria to decide issues like these.

Such issues have recently become important with new interest in digital multimedia libraries, or systems for information retrieval of multimedia data. Multimedia data can be so much costlier to fetch than text data because they can be so much larger; a poor method for retrieving them can take hours or days longer than a good method. So more analysis of a query is needed before data fetch, and finding the best way to retrieve is important. And typically, there is less explicit structure in a multimedia database than a relational database. For these reasons the best ways for multimedia retrieval tend to differ from the best ways for traditional database systems, as exemplified in [3, 5, 12, 22].

We find it helpful to use the concept of "information filters" [2] to discuss multimedia information retrieval. These processes take as input a set of data pointers, and return the subset that pass some necessary but not sufficient conditions for a data match. Different filters can work on separate parts of a query, on separate tests, or on separate media if each datum is multimedia (as when pictures have associated text captions or audio). We assume here that filters err only on the side of caution so that they never exclude relevant media objects (that is, they have perfect recall but imperfect precision). Even though detailed examination of the data would subsume their results, information filters can be cost-effective if their cost is significantly less than a full data match. But not all filters are cost-effective, nor all ways of using them.

Signature matching [4, 8, 10, 11] is a special case of information filtering that has been fruitfully applied first to text data and then to multimedia data. It extracts the key words in text, or the key shapes in pictures, or the key sounds in audio, and hashes them into a "signature table". At query time, query words or features are also hashed into the signature table. A hash hit on any word or feature is a necessary but not sufficient condition for an exact match between the query and some datum that was hashed there. The signature file can be stored in main memory, and using it can be considerably faster than searching a secondary-storage index to the data. Thus signature matching is a special case of information filtering as defined above, with the matching serving as a redundant but efficiency-improving filter. We would like to extend the notion of signature matching to consider multiple signatures for the same data items, perhaps representing different dimensions of the data, where any number of the signatures could be used to quickly rule out data items; and we could consider signatures of signatures. With a large set of possible signatures, it is not straightforward to decide which to trying matching to and in what order.

The primary objective of this paper is to present a general theory of optimal information retrieval with both nonredundant and redundant information filters. To accomplish this we provide important new results about signature and other redundant filters, which have not previously been carefully analyzed as abstracted components of an information-retrieval system. We will use a model of filter cost that corresponds closely to that of most information-retrieval implementations. We will first examine in sections 2-4 the most common kind of multifilter circumstance, conjunctive filtering, and provide simple local optimality conditions on a conjunctive sequence. The local optimality conditions concern interchanges of filter order, deletion of redundant filters, insertion of redundant filters, and concurrent execution of filters. We will prove that with a general cost model, most forms of concurrency are not desirable with conjunctive filtering, since the earlier starts of the concurrent filters do not compensate for the increased input they must handle. Section 4 will show results of experiments confirming the value of our local optimality criteria, and in particular that a simple "greedy" algorithm based on them has excellent average performance.

We then generalize our results to arbitrary boolean expressions involving filters in section 5. Disjunctive sequences are just the duals of conjunctive sequences, and negations are relatively straightforward in their optimality implications. Factoring of conjuncts over disjuncts and vice versa leads to an additional local optimality condition, but one that we argue is global in most applications.

One important application of signature matching is to supporting natural-language processing in information retrieval. Many researchers in information retrieval have pointed out the deficiencies of raw keyword matching (e.g. [17]), and parsing and semantic interpretation of natural-language data descriptions could be a solution. The main obstacle is speed. Depending on the approach, the required parsing, semantic interpretation, and semantic matching could take minutes where keyword matching requires seconds. But if we can decompose the required processing into several filters, we may be able to rule out most potential matches at an early stage without full natural-language processing. We discuss this in section 6 of this paper, and our theory permits us to improve the MARIE-1 system [24], which pioneered in the systematic use of natural-language captions on multimedia data as the primary indexing of the data.

Conjunctive information filtering

We first consider boolean queries with only conjunctions ("and"s). Such queries can be thought of as |m| information filters through which some data items must pass, where each data item must pass the test administered by each filter if it is to be part of the query answer. Let the event of passing filter |i| be termed |f sub i|. Assume each filter has an average cost of execution per data item of |c sub i|, and an a priori probability of passing a random data item of |p (f sub i)|, where |0 < p (f sub i) < 1| to avoid considering trivial cases. Generally the |c sub i| will be execution times so we can find the minimum execution time of a filter sequence, but our mathematics here applies to any costs. Assume further that costs are independent of probabilities, or that the cost of testing whether an item passes a filter is independent of the success or failure of the test or any other test on that data item; this is true of testing of uniform-size data by hash table lookups, for instance.

If the filters are applied in sequence, the expected total cost per data item will be: .EQ I (1) t sub 1,m = c sub 1 + c sub 2 p (f sub 1) + c sub 3 p (f sub 1 andsign f sub 2) + ... + c sub m p (f sub 1 andsign f sub 2 ... andsign f sub m-1) We would like to choose the filter sequence that minimizes |t sub i,m| for a set of |m| filters, or know if possible deletions of filters could improve cost. The |c sub i| and |p sub i| parameters can be estimated either from past statistics of the filters on similar problems, or by applying the filters to a small random sample of the database.

Using this formula directly to compute the best of the |m !| possible permutations of a set of |m| filters would require estimating |m| average filter costs and |m !| probabilities. Then evaluating each proposed sequence requires |m - 1| additions and |m - 1| multiplications, for a total of |m ! (m - 1)| additions and |m ! (m - 1)| multiplications. This is considerable work for even small sets of filters. And we have not yet considered that some of the filters could be redundant and deletable. So we seek mathematically justifiable shortcuts.

Related work

Related problems to finding optimal conjunctive sequences have been examined elsewhere. In the database literature this is the problem of restriction-order optimization based on selectivities (called "single-variable" optimization in [16]), but the focus there has been on solving the more critical problem of optimizing joins and other operations that generally are far worse bottlenecks in processing time for databases. The usual methods for restriction-order optimization require search, either exhaustive or heuristic, in the space of possible rearrangements of a query expression, rather than attempting to find general optimization criteria. Work in

semantic query optimization for database queries sometimes suggests signature-table methods [26], though it is usually concerned with application of more-complicated "integrity constraints".

Some work in automatic testing of electronic and mechanical devices has considered optimal test ordering. If we know the possible faults in a device and their probabilistic distribution, and we know a many-to-many mapping between test results and faults, the optimal test sequence can be found by an AO* search [20]. While the tests can be considered conjunctive filters where fault sets are the data items, these filters are far simpler and there are far more of them than in the applications we will consider here. Thus, statistical-distribution assumptions [18] are usually made which we prefer to avoid.

For optimization of rule-based systems, [28] analyzed optimization of filter sequences that create persistent variable bindings, a different problem but related to ours. Work in Markovian decision processes [19] has developed general methods for situations more complicated than sequences, but these are not very efficient for sequences. Work on optimal decision trees generally assumes all costs terms are equal, which leads to specialized algorithms. Psychological work on "sequential decision-making" and the associated "sequential ordering problem" is actually unrelated, as these terms describe psychological modeling of pattern inference from sequences with no interest in finding an optimal way to do it. Work on "linear placement" in VLSI design is unrelated because the concern is generally with geometric relationships and with minimizing chip size by minimizing interconnections; our problem assumes negligible interconnection cost.

Problems of task scheduling that are related to conjunctive-sequence ordering are generally NP-complete [9] because generally we must examine some constant fraction of all possible sequences in order to find the optimal one. But in this paper, we will propose some quick polynomial-time criteria that can be used to rule out all but a few possible sequences, as for instance criteria that sort the sequence. While these criteria are not guaranteed to find the optimal solution, they usually do, as we have confirmed by experiments, and furthermore they usually greatly improve the average-case execution time of the sequence.

Local criteria for interchange-optimality of the cost of a conjunctive filter sequence

We would like to find filter sequences that are optimal with respect to cost among neighboring filter sequences ("locally optimal"). By "neighboring" we shall mean sequences creatable by interchanging a few adjacent filters, deleting a few filters, inserting a few filters, or some combination thereof.

First, consider the effect on cost of interchanging filters in a conjunctive sequence. If a sequence is a local optimum, then any such interchange must not decrease the total cost. Consider the effect of interchanging adjacent filters |i| and |i+1|. This certainly cannot affect the cost terms for filters before |i|, and it cannot affect the cost terms for filters after |i+1| because |f sub 1 andsign f sub 2 = f sub 2 andsign f sub 1|. So the interchange of filters |i| and |i+1| will not improve cost if:

```
c sub i p ( f sub 1 andsign f sub 2 andsign ... f sub i-1 )   +
c sub i+1 p ( f sub 1 andsign f sub 2 andsign ... f sub i-1 andsign f sub i )    <=
c sub i+1 p ( f sub 1 andsign f sub 2 andsign ... f sub i-1 )   +
c sub i p ( f sub 1 andsign f sub 2 andsign ... f sub i-1 andsign f sub i+1 )
```

or, after converting to conditional probabilities:

```
c sub i   +
c sub i+1 p ( f sub i | f sub 1 andsign f sub 2 andsign ... f sub i-1 )    <=
c sub i+1   +
c sub i p ( f sub i+1 | f sub 1 andsign f sub 2 andsign ... f sub i-1 )
```

we then get after rearranging: .EQ I (2) c sub i / [1 - p (f sub i | f sub 1 andsign f sub 2 andsign ... f sub i-1)] <= c sub i+1 / [1 - p (f sub i+1 | f sub 1 andsign f sub 2 andsign ... f sub i-1)] In other words, a locally optimal sequence must be sorted by |c / q|, where |q| is the fraction of the items failing a filter after passing all previous filters. We call this "interchange optimality".

For example, suppose we have three filters with costs |c sub 1 = 10|, |c sub 2 = 24|, and |c sub 3 = 20|. Suppose |p (f sub 1) = 0.5|, |p (f sub 2) = 0.4|, |p (f sub 2 | f sub 1) = 0.6|, and |p (f sub 3 | f sub 1) = 0.8|. Then |c sub 1 / (1 - p (f sub 1)) = 20|, |c sub 2 / (1 - p (f sub 2)) = 40|, |c sub 2 / (1 - p (f sub 2 | f sub 1)) = 60|, and |c sub 3 / (1 - p (f sub 3 | f sub 1)) = 100|. Thus the filter order 1-2-3 is better than the order 2-1-3 because |20 < 40|, and 1-2-3 is better than 1-3-2 because |60 < 100| (despite the fact |c sub 3 < c sub 2|).

If we can assume that filter-acceptance events |f sub i| and |f sub i+1| are independent of all the previous filter-acceptance events, the conditional probabilities become the a priori probabilities |p (f sub i)| of a random data item passing a filter |i|, and we get (Theorem 1 of [15]): .EQ I (3) c sub i / (1 - p (f sub i)) <= c sub i+1 / (1 - p (f sub i+1))

Entailing and entailed filters

Unfortunately, we cannot often assume independence of filter pairs because signature matching is not independent of full matching. Let us first define entailment:

```
"Definition 2.1: filter"    f sub i    "entails filter"    f sub j
"if and only if"    p ( f sub j | f sub i ) = 1
```

We assume that entailment is always absolute if it occurs at all; in other words for any filter, |p (f sub j | f sub i)| is either |p (f sub j)| (independence) or 1 (entailment). Filters usually can be designed to accomplish this (this is the point of signatures and hash functions), though it should be noted that if |f sub 3| entails |f sub 1|, and |f sub 3| entails |f sub 2|, then |f sub 1| and |f sub 2| cannot be completely independent, although they could very near independence. A signature filter is an entailed filter.

We would like to cover some complex entailment networks by our theory. So we will allow that a filter can entail more than one other filter. However, to prevent confusion we assume that a filter can be entailed by only one other, so if A entails B and B entails C, we assume A does not directly entail C unless B is deleted.

When filter |e| is in a sequence where it entails some previous filter, |p (f sub e | u) != p (f sub e)| where |u| represents the event of passing all the filters before |e|. But we can use Bayes' Rule. Suppose |u| can be broken into two pieces such that |u = u sub 1 andsign u sub 2| where |u sub 1| are the filters entailed by |f sub e| (so |u sub 2| are the filters independent of |f sub e|). Then: .EQ I (4) p (f sub e | u) = p (f sub e | u sub 1) = p (f sub e andsign u sub 1) / p (u sub 1) = p (f sub e) / p (u sub 1) To obtain |p (u sub 1)| if all the filters in |u sub 1| are independent, multiply their probabilities. Otherwise, eliminate the filters that are entailed by others in |u sub 1| and use conditional probabilities as necessary to figure the total probability of the rest.

For instance, if filter 7 entails filters 3 and 4 but is independent of filters 1, 2, 5, and 6, and the a-priori probability of passing filter 3 is 0.4, of passing filter 4 is 0.5, of passing filter 7 is 0.1, and filter 3 is near-independent of filter 4, |p (f sub 7 | f sub 1 ... andsign f sub 6) approx 0.1 / (0.4 * 0.5) = 0.5|. If filter 7 is then followed by a filter 8 of equal cost but with an independent success probability of 0.6, filters 7 and 8 should not be interchanged in search of local optimality since |c / (1 - 0.5) < c / (1 - 0.6)|.

Another consequence of our assumption of absolute-or-none entailment between filters is that |p (f sub i | u) != (f sub i)| only when |i| is entailing, and then it is only a function of the set of entailed filters. But an entailed filter

must precede its entailing filter in a sequence to be useful. So |p (f sub i | u)| will be a constant for all useful placements of an entailing filter |f sub i| in a given filter sequence. Then inequalities (2) and (3) are like sorting criteria for a bubble sort on the filter sequence. If we bubble-sort using interchange optimality and the resulting sequence has each entailed filter preceding its entailing filter, we have found the optimal order for the sequence in polynomial time with respect to the number of filters. If the sort result does not obey entailment relationships, we must try something else that is likely not polynomial; section 4.2 gives an algorithm.

Note that even if entailment is not all-or-none, inequality (2) is still a necessary condition on a locally or globally optimal filter sequence. Inequality (2) may then still be used as a sorting criterion to obtain a locally optimal sequence provided we can guess which entailed filters will be necessary. But guessing does not give a polynomial-time method in the worst case.

Deleting an entailed filter

With entailment, there is a new possibility for improving the cost of a set of filters without changing the answers they produce: deletion of an entailed filter. Assume the entailed (fast) filter is |i| and its entailing (slow) filter is |e|. The deletion of |i| cannot affect the cost terms for filters before |i|. It cannot either affect the cost terms after |e| because any data item |i| removed will now be removed by |e|. So deletion of filter |i| does not improve cost if:

```
c sub i p ( f sub 1 andsign ... f sub i-1 ) +
c sub i+1 p ( f sub 1 andsign ... f sub i-1 andsign f sub i ) +
c sub i+2 p ( f sub 1 andsign ... f sub i-1 andsign f sub i andsign f sub i+1 )
+ ... +
c sub e p ( f sub 1 andsign ... f sub e-1 )

  <=
c sub i+1 p ( f sub 1 andsign ... f sub i-1 ) +
c sub i+2 p ( f sub 1 andsign ... f sub i-1 andsign f sub i+1 )
+ ... +
c sub e p ( f sub 1 andsign ... f sub i-1 andsign f sub i+1 ... f sub e-1 )
```

or after introducing conditional probabilities: .EQ I (5) c sub i + c sub i+1 p (f sub i | f sub 1 ... andsign f sub i-1) + c sub i+2 p (f sub i andsign f sub i+1 | f sub 1 ... andsign f sub i-1) + ... + c sub e p (f sub i ... andsign f sub e-1 | f sub 1 andsign ... f sub i-1)

```
  <=
c sub i+1 + c sub i+2 p ( f sub i+1 | f sub 1 andsign ... f sub i-1 )
+ ... +
c sub e p ( f sub i+1 ... andsign f sub e-1 | f sub 1 andsign ... f sub i-1 )
```

We call this condition "deletion optimality".

If we can assume that the probability of passing entailed filter |i| is independent of passing all other filters |f sub j| from |j = 1| to |e-1| we get after rearrangement: .EQ I (6) c sub i / (1 - p (f sub i)) <= c sub i+1 + c sub i+2 p (f sub i+1 | f sub 1 andsign ... f sub i-1) + ... + c sub e p (f sub i+1 ... andsign f sub e-1 | f sub 1 andsign ... f sub i-1) Note that the right side is just |t sub i+1,e| in the notation of equation (1), or the expected cost of the filter sequence |f sub i+1|, |f sub i+2|, ..., |f sub e| alone.

As an example, suppose we have three filters where filter 1 is entailed by filter 3, and filter 2 is independent of the others. Suppose |c sub 1 = 10|, |c sub 2 = 24|, |c sub 3 = 20|, |p (f sub 1) = 0.5|, and |p (f sub 2 | f sub 1) = 0.4|. Then |c sub 1 / (1 - p (f sub 1)) = 20| and |c sub 2 + c sub 3 p (f sub 2 | f sub 1) = 32|. Hence sequence 1-2-3 is preferable to sequence 2-3; that is, redundant filter 1 should not be deleted from the sequence 1-2-3.

Another way to simplify (5) is to note that |p ((f sub j andsign r) | f sub 1 andsign ... f sub j-1) <= p (r | f sub 1 andsign ... f sub j-1)|, and we can use this to match each pair of the last |e - i - 1| terms on the left side and the right side. Then eliminating each pair, we get a simpler sufficient condition for (5) to be true: .EQ I (7) c sub i / [1 - p (f sub i | f sub 1 andsign ... f sub i-1)] <= c sub i+1 Note that condition (7) for filter |i| implies the interchange optimality condition (3) for filters |i| and |i+1|, since |c sub i+1 < c sub i+1 / (1 - p (f sub i+1 | f sub 1 andsign ... f sub i-1))|.

Deletion of more than one entailed filter

A question arises about deletion optimality: Even if filters |i| and |j| are individually deletion-optimal in a particular filter sequence, could the deletion of *both* of the them be locally optimal? The following definitions will provide the criteria for such stronger forms of optimality. The idea behind them is to ensure that local interchange and deletion optimality will always hold between the remaining filters no matter which intervening entailed filters are deleted.

> *Definition 2.2: A filter that is not entailed is "strongly-deletion-optimal". An entailed filter |i| is strongly-deletion-optimal if it is deletion-optimal by condition (7), and also |c sub i+1 <= c sub j| for all |j| such that |i < j <= r|, where |r| is the next non-entailed filter after |i| if any, else |r| is the last filter.*
>
> *Definition 2.3: A filter that is not entailing is "strongly-interchange-optimal" if it is interchange-optimal, condition (2), with respect to its two neighbors. An entailing filter |i| is strongly-interchange-optimal if it is interchange-optimal and |c sub j / (1 - p (f sub j | f sub 1 ... andsign f sub j-1)) | |< c sub i / (1 - p (f sub i | f sub 1 ... andsign f sub j-1))|, for all |r <= j < i| where |f sub j| is not entailed by |f sub i| and where |r| is the last filter before |i| that is not entailed by |i| if any, else |r| is the first filter.*

As an example, consider the filter sequence 1-2-3-4 where where filter 1 is entailed by filter 4, and filter 2 is entailed by filter 3. Suppose |c sub 1 = 10|, |c sub 2 = 20|, |c sub 3 = 40|, and |c sub 4 = 30|. Suppose |p (f sub 1) = 0.5|, |p (f sub 2) = 0.4|, |p (f sub 2 | f sub 1) = 0.5|, |p (f sub 3 | f sub 1) = 0.8|, |p (f sub 3 | f sub 1 andsign f sub 2) = 0.85|, |p (f sub 4 | f sub 1 andsign f sub 2) = 0.9|. Then ordinary deletion optimality by sufficient condition (7) holds for filter 1 since |20 <= 20|, and for filter 2 since |40 <= 40|. Strong deletion optimality just requires one additional condition, for filter 1, that |c sub 2 < c sub 3|, which holds since |20 < 40|. Ordinary interchange optimality by inequality (2) holds for the sequence 1-2-3-4 since |20 < 33.3|, |40 < 200|, and |267 < 300|. Strong interchange optimality just requires one additional condition, for filter 3 to cover the case of filter 2 being deleted, that |c sub 1 / (1 - p (f sub 1)) < c sub 3 / (1 - p (f sub 3))|, which holds no matter what |p (f sub 3)| is since |10 / (1 - 0.5) = 20 < 40|. Hence the filters in the sequence 1-2-3-4 are strongly-deletion-optimal and strongly-interchange-optimal. This means not only that one of filters 1 and 2 should not be deleted, but that deleting filter 1 does not affect the desirability of deleting filter 2 and vice versa, as the following Lemma will show.

Lemma 2.1, Subsequence Deletion Suboptimality Lemma: Given a sequence S of filters in which for every filter pair (not necessarily adjacent filters), the two filters are either probabilistically independent or else one filter entails the other. Suppose entailed filter |i| in S is strongly-deletion-optimal. Then it remains strongly-deletion-optimal for any subsequence created by deleting some entailed filters of S.. Proof: The |c sub i+1 <= c sub j| condition in the strong deletion optimality definition must remain true when filters are deleted because filter order must be maintained and "the next nonentailed filter" cannot be deleted. So we only need show that condition (7) holds for sequences resulting from deletion from S. Filters deleted from S that occur after |i+1| cannot affect either side of (7), so they can be ignored. Filters deleted that are independent of filter |i| cannot

affect either side of (7) either. If a filter |j| entailed by filter |i| is deleted (|j < i| to make sense), and |u| represents the event of passing the remaining filters before |i|,

```
p ( f sub i | u ) = p ( f sub i andsign u ) / p ( u ) < p ( f sub i andsign u ) / p ( u
andsign f sub j ) = p ( f sub i andsign u andsign f sub j ) / p ( u
andsign f sub j ) = p ( f sub i | u andsign f sub j )
```

since |f sub i = f sub i andsign f sub j|, so (7) still holds. The only remaining case is when filter |i+1| is among those deleted, and some filter |j| originally to the right of it immediately follows |i| after all deletions. Then by the definition of strong deletion optimality, |c sub i+1 < c sub j| for any filter |j| that could become the next filter after |i| by deletions, and (7) holds for the new filter. We only need to consider filters up to the next nonentailed filter, because that one cannot be deleted. QED.

We can use this lemma to get sufficient conditions to say that a filter sequence is the globally optimal one with respect to interchanges and deletions. The conditions require only polynomial time to confirm. This result applies to an important class of problems, and filters can be purposely designed to make global optimality easier to guarantee.

Theorem 2.1, Restricted Global-Optimality Theorem: Given a set of filters in which any two filters are either probabilistically independent or else one of the two entails the other. Assume in some sequence S of those filters that every filter is strongly-interchange-optimal, and every entailed filter is strongly-deletion-optimal. Then S is the global optimum in the space of improper subsequences created by deletions from it and/or permutations of it.
Proof: By Lemma 1, any subsequence T created solely by deletions (with no permutations) must also be strongly-deletion-optimal. Since each T can be created by a single deletion from another strongly-deletion-optimal sequence, it must cost more than that longer sequence because strong deletion optimality implies deletion optimality. Hence by transitivity, T must be more costly than S.

Now we show that permutation of a subsequence T could not improve its cost. For this, we need only consider moving of an entailing filter |e| because entailing filters are the only ones whose interchange optimality is affected by deletions, and they are only affected by deletions of the filters they entail, according to the assumptions of section 2.3. The ratio |c / (1 - p)| for filter |e| will be decreased by deletions of filters left of it, so we might think we would need to move it left to restore sorted order on |c / (1 - p)|. However, if strong-interchange-optimality holds, it is never desirable to interchange |e| with a filter it entails at |e - 1|. Similarly, if we delete a filter at |e - 1| that is entailed by |e|, strong-interchange-optimality says that it is never desirable to interchange |e| with |e - 2|. We can continue to filter |r|, the last filter before |e| that is not entailed by |e|; filters before |r| cannot be placed next to |e| because |r| cannot be deleted. Hence it cannot improve cost to move filter |e| after deleting filters left of it. QED.

Note two important cases to which the Theorem applies: if there are no entailed filters in a sequence, or if one entailing filter entails all the others (as in "multi-level" signature matching [4]). The Theorem can also usefully rule out subsequences of the original filter sequence even when it cannot apply to the original sequence.

Assuming that all conditional probabilities are known in advance, checking strong-deletion-optimality by Theorem 2.1 for |m| filters requires |O (m)| comparisons and |O (m)| subtractions and divisions, and checking strong-interchange-optimality requires |O (m sup 2)| comparisons and |O (m sup 2)| subtractions and divisions. If the conditions are passed, a worst-case exponential number of possible subsequences are eliminated from consideration. So a heuristic for finding an optimal sequence for a set of filters is to interchange-sort the full set of filters and check for correct entailment orders, strong-deletion optimality, and strong-interchange-optimality; if they hold, you have the global optimum in polynomial time. Otherwise, the global optimum must have at least one filter deleted.

Inserting entailed filters

Another way to improve the cost of a sequence is to do the opposite of deleting a filter, inserting an entailed filter somewhere before its entailing filter in the sequence. (It makes no sense to insert or delete a nonentailed filter, as without such a filter, the final output of the filter sequence must be wrong.) The appropriate criteria are just opposite of inequalities (5), (6), and (7). Insertions can be ignored if we start with the set of all filters and consider all possible deletions to it; this will be the strategy of the algorithms in section 4.

Distributed and parallel filtering

So far we have only considered sequential implementation of a conjunctive filter sequence. A distributed implementation could speed processing considerably. One approach is to assign each filter to a processor, send each filter processor each data item, and have all the processors send their output to a single "intersection processor" that finds items that pass more than a threshold number of filters. This is useful for keyword matching where each processor gets a keyword. Disadvantages are that the intersection processor can be a bottleneck (although its input will generally arrive irregularly, evening the workload), and the degree of parallelism is limited by the number of filters which can be devised.

Another approach is to assign subsets of data items to processors, as in the work [30] on the Connection Machine, a massively parallel machine. Keywords in sequence were supplied to all processors simultaneously, and each processor counted the number of matches for each data item. Then processors were polled to get all data items with more than a threshold number of matches. This approach is easy to implement on the right hardware, and can get answers very fast, with a speedup close to linear with the number of processors used. But this massive parallelism also means massive idleness: Usually most data items do not match the query, and their processors just sit uselessly. So this approach is very wasteful of computer resources, something which is hard to justify for a non-critical application like information retrieval and a multi-million dollar massively-parallel machine. So we will will explore here partition of the data items with a significantly lower degree of parallelism, an approach that could work for, say, networked workstations.

Data-partition parallelism

Suppose we have |N| processors for the filters, where each processor filters a randomly chosen disjoint partition of the data items. Each would first apply filter 1 to its partition, then filter 2 to the output of filter 1, etc. Assume that the cost of applying a filter to a set of data items is proportional to the number of data items. This is true for most filters since most useful filtering methods do not require examining the interaction of data items, and each filter can easily be supplied with a random data subset; it is true for signature table methods as well as the natural-language processing filters of section 6. Then without overhead, filters will take in |N| times less time with |N| processors. Otherwise, assume that overhead is |k sub 0 + k sub 1 N| where |k sub 0| and |k sub 1| are constants. The |k sub 0| covers any initialization done by a central process that is independent of the number of filtering processes (like sending the data items to the processes), and the |k sub 1| covers data-independent initialization for each filtering process (like process-creation software). This model is a good approximation for much simple multiprocess software, like the Quintus Prolog TCP utility for Unix that we used in the experiments reported in section 6.

So the total cost of doing a sequence of filters with data-partition parallelism is |k sub 0 + k sub 1 N + (c sub 1 / N) + (p (f sub 1) c sub 2 / N) + (p (f sub 2 | f sub 1) c sub 3 / N) + ... |, which has a minimum with respect to |N| at: .EQ I (8) N sub best = sqrt { (1 / k sub 1) (c sub 1 + p (f sub 1) c sub 2 + p (f sub 2 | f sub 1) c sub 3 + ...) } = sqrt { t sub i,m / k sub 1 } This must be rounded to an integer. Usually this is larger than |N| since |k

sub 1| is usually much less than the filter costs. Then since the derivative of the cost is negative for |N < N sub best|, the best we can do is to use all |N| processors. This is only different with hardware that poorly supports multiple processes, for filters that are all very simple (in which case some ought to be coalesced), or very large numbers of processors (something difficult to justify for information retrieval). And with a sequence of filters, the setup cost need only be incurred for the sequence once because each processor applies the data to each filter in turn, so its cost can be amortized; and the final union of results is just a disjoint union, easy to accomplish.

But the processors need not do the same things; subsets of the processors might have exclusive responsibility for certain filters. We can prove this is never desirable under a few simple assumptions. We can even prove it for a more general processing-cost model than the preceding one, where the cost of a filter is |g (N) + (c sub i / N)| where |N| is the number of processors for |g (N)| linear or concave (since there ought to be economies of scale in invoking large numbers of processors).

Theorem 3.1, Parallel-filter Theorem: Suppose we have |N| processors to implement two information filters. Suppose the cost, per unit number of data items, of |n| processors doing a filter |i| is |g (n) + (c sub i / n)|, |g (n)| the overhead cost, and |c sub i| the cost of the filter per data item as above. Assume |g prime prime (n) <= 0| and |g (0) = 0|. Then it is best to apply all |N| processors to one filter, then all |N| processors to the other filter. Proof: Suppose we do assign |n sub 1| processors to filter 1 and |N - n sub 1| processors to filter 2, for some |n sub 1 <= N|. Then execution time for just the two filters plus overhead will be |g (n sub 1) + g (N - n sub 1) + max ((c sub 1 / n sub 1) , (c sub 2 / (N - n sub 1)))|. Now |(partial / partial n sub 1) (c sub 1 / n sub 1) < 0| and |(partial / partial n sub 1) (c sub 2 / (N - n sub 1)) > 0|. Hence the minimum of the |max| term will be when |c sub 1 / n sub 1 = c sub 2 / (N - n sub 1)|, or when |n sub 1 = N c sub 1 / (c sub 1 + c sub 2)|, at which value the cost attains a minimum of |g (N c sub 1 / (c sub 1 + c sub 2)) + g (N c sub 2 / (c sub 1 + c sub 2)) + ((c sub 1 + c sub 2) / N)|.

On the other hand, if all |N| processors are assigned to perform filtering operation 1, then all to filtering operation 2, the cost of the two filters plus overhead will be |g (N) + (c sub 1 / N) + (p sub 1 c sub 2 / N)| where |p sub 1| is the probability of passing filter 1. We will prove now that |g (N c sub 1 / (c sub 1 + c sub 2)) + g (N c sub 2 / (c sub 1 + c sub 2)) + ((c sub 1 + c sub 2) / N)| | >= g (N) + (c sub 1 / N) + (p sub 1 c sub 2 / N)|. We will do this using the rule that if |a >= b| and |c >= d|, then |a + c > b + d|. It is easy to see that |(c sub 1 + c sub 2) / N >= (c sub 1 / N) + (p sub 1 c sub 2 / N)| because that inequality simplifies to |1 >= p sub 1|, which is true for nontrivial filters. So it is sufficient to show that |g (N c sub 1 / (c sub 1 + c sub 2)) + g (N c sub 2 / (c sub 1 + c sub 2)) >= g (N)|. If we use again |n sub 1 = N c sub 1 / (c sub 1 + c sub 2)|, we can rewrite the inequality more simply as |g (n sub 1) + g (N - n sub 1) >= g (N)|. Since |g (0) = 0|, this can be rewritten as |[g (n sub 1) + g (N - n sub 1)] / 2 >= [g (0) + g (N)] / 2|. In graphical terms, we are asking whether the chord across |g (x)| between abscissas |n sub 1| and |N - n sub 1 | is higher at its midpoint at abscissa |N / 2| than the chord across |g (x)| between |0| and |N| is at its midpoint at the same abscissa |N / 2|. But |g prime prime <= 0|, so the left end of the first chord cannot lie below the second chord; and the right end of the first chord cannot lie below the second chord. And chords are straight, so they cannot intersect more than once; so the entirety of the first chord cannot lie below the entirety of the second chord. Hence |g (n sub 1) + g (N - n sub 1) >= g (N)|. Hence assigning all |N| processors to do filter 1, then all |N| processors to do filter 2, is superior to doing both filters in parallel.

The above analysis is actually conservative. To implement parallel filtering, we must round from |n sub 1| to an integer, and this worsens the cost further. Furthermore, the above results assume that with sequential filtering, all |N| processors do filter 1, then all |N| do filter 2, etc. But if, say, processor 6 finishes filter 1 early, it could start applying filter 2 to its results of filter 1, before processor 5 finishes filter 1. This interleaving effect is data-dependent and hard to analyze, but could improve processing speed by using otherwise-idle time. QED.

The following theorem generalizes this result on two filters to arbitrary execution plans involving parallelism for a set of filters.

Theorem 3.2: Given an execution plan for a set of filters on N processors, expressed as a directed acyclic graph where each node has an associated filter and one of N subdivisions of the data to which it applies. Suppose the cost, per unit number of data items, of |n| processors doing a filter |i| is |g (n) + (c sub i / n)|, and |c sub i| the cost of the filter per data item; and assume |g prime prime (n) <= 0| and |g (0) = 0|. Then if that execution plan has work on different filters in parallel, it is not optimal. Proof: Such an execution plan could be transformed into a sequence of filters by repeatedly taking a pair of parallel strands and either (a) coalescing them to a single filter on a bigger set of data items, if they perform the same filter operation, or (b) sequencing them arbitrarily otherwise. Then if we reverse the order of these transformations, we will get the original execution sequence. But in this latter process, Theorem 3.1 applies at every step, so the original parallel execution plan cannot be optimal even if the completely sequenced plan is optimal. Furthermore, if a filter appeared more than once in the original execution plan, it will appear more than once in the completely sequenced plan, so that plan cannot be optimal anyway. QED.

As an example of Theorem 3.2, |f sub 1| in parallel with the sequence of |f sub 2| followed by |f sub 3| cannot be optimal because the sequence |f sub 1 - f sub 2 - f sub 3| would be better by Theorem 2.2. Similarly, suppose we do |f sub 1| then |f sub 3| on one parallel track, |f sub 2| then |f sub 3| on another, where the starting time of |f sub 3| on the first track could be different than the starting time on the second; then any sequencing would have each filter once plus |f sub 3| twice, which is worse than just having each filter once.

Parallel non-filtering processes

Information-filtering applications can require additional non-filtering processing. In natural-language information retrieval, for instance, parsing and interpretation of the natural language is necessary before detailed matching can be done. The effect of such processes can be modeled as imposing earliest start times for all the processes that depend on them. This introduces additional inequality constraints of a more traditional sort into our scheduling problem.

We can handle these constraints using the standard method of optimization with linear inequality constraints using active sets, as in section 5.2 of [13]. If the local optimality conditions can be satisfied without violating the new start-time constraints, the local optimum remains a local optimum. Otherwise, the local optima must be on the border of the region of feasibility with the minimum number of "active" constraints (inequalities reducing to equalities). That means that any local optimum of the new problem must satisfy interchange optimality and local deletion optimality except in a minimum number of places necessary to satisfy the start-time constraint.

Algorithms for conjunctive filter-sequence optimization and experiments on them

Some useful results

First we must define a canonical form for a filter sequence prior to considering a deletion from it.

> *Definition 4.1:* A conjunctive filter sequence is interchange-entailment sorted if all entailed filters precede their entailing filters, and it obeys interchange optimality except possibly where an entailed filter is immediately followed by its entailing filter.

An example of the latter clause would be the sequence of two filters where filter 2 entails filter 1 and |c sub 1 / (1 - p (f sub 1)) > c sub 2 / (1 - p (f sub 2))|.

Theorem 4.1: For a given set of filters, there is only one interchange-entailment sorted sequence, exclusive of ties on interchange desirability, and it can be obtained by a bubble sort. Proof: If there were more than one such sequence, both must have interchange optimality for all filter pairs except between an entailed and entailing filter. If both sequences had the same filters just after the entailed filters, then all the other filters would have to precede those entailed filters in interchange-sorted order, and the two sequences would have to be identical. So assume there is at least one entailed filter |e| that is immediately preceded in sequence 1 by filter |i| and in sequence 2 by filter |j|, |i != j|. Assuming no ties, one of |i| and |j| must have a larger ratio |c / (1 - p)|, so assume |i| is the larger. Then sequence 2 must have |i| somewhere before |j| since |e| entails both |i| and |j|. But then the sequence |i| is not interchange-optimal because |i| could be interchanged for improved sequence cost with a filters to its right, or with |j| if no other. Hence there can be only one interchange-entailment sorted sequence.

The bubble sort referred to can use the following sorting criteria:

--If filter |i+1| is right of its entailing filter, interchange filter |i| with filter |i+1|.
--Otherwise, if |r sub i > r sub i+1|, interchange filter |i| with filter |i+1|.

Since both of these criteria make progress toward a interchange-entailed sorted sequence at every interchange, and they are applied repeatedly until no more changes need be made, they should eventually sort the sequence. QED.

The following useful theorem finds the opposite of strong-deletion-optimality, identifying certain filters that should be deleted no matter what is deleted around them.

Theorem 4.2: If a conjunctive filter sequence is interchange-entailment sorted, and an entailing filter |e| is immediately preceded by one of its entailed filters, a filter that is not itself entailing, and |c sub e-1 / (1 - p (f sub e-1)) > c sub e / (1 - p (f sub e | u))| where |u| is everything entailed by |e|, then filter |e-1| should be deleted no matter what other filters in the sequence are deleted. Proof: If |e-1| is not entailing, a sufficient condition (7) for deletability of |e-1| is that |c sub e-1 / (1 - p (f sub e-1)) > c sub e|. But that follows from transitivity on the inequality we already know and |c sub e / (1 - p (f sub e | u)) > c sub e|. QED.

A general algorithm for conjunctive filter-sequence optimization

Using the preceding results, we can now provide a general method for filter-sequence optimization.

1. For each filter in set S, compute |r sub i = c sub i / (1 - p (f sub i | v sub i))| where |v sub i| is the subset of S entailed by filter |i| if any.

2. Do an interchange-entailment sort of S.

3. Delete from S all filters satisfying the conditions of Theorem 4.2.

4. Initialize an agenda to hold this one sequence, and conduct a best-first search:

   (a) Remove the minimum-cost sequence from the agenda.

   (b) If Theorem 2.1 applies to this sequence, and it is also interchange-optimal, move it to a "potential answer" array and do not generate successors. (Such a sequence must be

eventually found, since the sequence without any deletable non-strongly-deletion-optimal filters will satisfy the conditions.)

(c) Otherwise, add all possible unexamined "successors" of this sequence to the agenda, together with their costs. A successor is obtained by deleting an entailed filter that is not strongly-deletion-optimal, recomputing the |r sub i| of its entailing filter, then redoing the sort.

(d) Return to (a).

5. When no agenda remains, find the lowest-cost potential answer sequence. For |N| processors available, assign |min (N , N sub best)| to each filter in that sequence.

This algorithm has exponential time complexity in the number of filters because of the possibly exponential number of combinations that must be considered in step 3, but it is simple to implement and handles any set of filters.

Experiments with random data

Analysis of filter execution plans can vary greatly in difficulty depending on the parameters of the filters involved. To better judge the number of local optima and how often the global optimum is easy to find, we conducted experiments with randomly generated filters. Given a particular number of filters to create, we randomly designated certain ones as entailing filters, and randomly chose some filters for them to entail. Entailment relationships were restricted to form a forest, following the discussion of section 2.3. (The forest restriction does allow one filter to entail two; for instance, |f sub D| could be a full match to the data item, entailed filter |f sub B| a match to its high-order bits, and entailed filter |f sub C| a match to its low-order bits.) Note that deletion of a node from a forest and rerouting the children nodes maintains the forest property.

Costs were randomly assigned to each filter from the uniform distribution 0 to 10. Filter success probabilities were assigned from the uniform distribution 0.01 to 0.99; for nonentailing filters, this was taken as the a priori probability, and for entailing filters, this was taken as the conditional probability given that all previous filters succeeded.

Fig. 1 shows just one example with four randomly generated filters. Their parameters are listed in the first four lines: The first argument to "filter" is the filter number, the second its average cost, and the third its probability information. "Independent" means the filter does not entail any others, and "Dependent" means it entails the filters whose numbers are listed, so filter 2 entails 1, and filter 3 entails 2. For these filters, there are four sequences of filter sets and subsets that satisfy interchange optimality and the entailments. Of these, two satisfy deletion optimality criteria and are marked as locally optimal (although in our experiments, it was far more usual to find only one). The first optimum is the global optimum, and the heuristic greedy algorithm to be discussed in section 4.4 finds it. Note how deletion interacts with interchange optimality: Filter 4 should precede filter 3 when filters 1 and 2 are present, but when 1 and 2 are deleted, filter 3 becomes more valuable and must precede 4.

Fig. 2 tabulates experimental results from a Quintus Prolog implementation which are graphically represented in Figs. 3-11. Each row of Fig. 2 summarizes 1000 randomly generated filter sets. The first column is the number of filters, the second the probability that a filter would be selected for possible entailment (not counting the times an entailment was ruled out because it would violate the forest property), and the third the probability that a filter is entailing (or actually, whether we should attempt to construct entailment relationships for it). The remaining columns show experimental results as means of natural logarithms, with associated standard errors in

parentheses; we use logarithms because they are better at summarizing combinatorial experiments. The fourth column is the mean of the logarithms of the number of possible subsequences that need to be considered, after interchange-entailment sorting, for each set of randomly generated filters. The fifth column is the mean of the logarithms of the number of those subsequences that were judged locally optimal with respect to the criteria of section 2, a number generally considerably smaller than that of the previous column. Note the values in the fifth column increase more slowly than the values in the fourth column.

Fig. 3 plots the size of the search space, the total number of sequences |n !| for |n| filters, with the values displayed in fourth column of Fig. 2; four assignments of entailment parameters are shown. Figs. 4-7 show these last four curves plotted against the number of locally optimal sequences, the fifth column of Fig. 2.

A greedy algorithm for filter-sequence optimization

The sixth column of Fig. 2 shows the performance of a simple heuristic "greedy" algorithm to find the optimum, in terms of the means of the logarithms of the ratios of the cost of the sequence found by the algorithm to the cost of the true optimum sequence. This algorithm performs far better than the algorithm of section 4.2 with random filters. At each step, it deletes the best filter that it can (the entailed filter whose deletion followed by resorting most improves overall cost), and does an interchange-entailment sort again, until no further deletion can improve cost. No backtracking is done. This greedy algorithm is |O (m sup 3)|, |m| the number of filters, since sorting is |O (m sup 2)|; there are |O (m)| things to delete and hence |O (m)| steps; each step looks at |O (m)| subsequences and evaluates the cost of each subsequence in |O (m)| time, then resorts in |O (m)| time to reposition one entailing filter. The greedy algorithm cannot get the optimal solution all the time. But the sixth column demonstrates that it nearly always gets the correct answer for up to fifteen filters, and its rate of deterioration is considerably slower than the increases in the size of the problem space, the number of sequences considered, and the number of local optima. Figs. 8-11 plot columns 5 and 6 of Fig. 2 against one another.

General boolean filters

We now extend the previous analysis to queries or filter execution plans that are equivalent to arbitrary boolean expressions ("general boolean filters"), with the inclusion of disjunctions and negations of filters.

Ordering of disjunctions

The idea of filters in disjunction is that if the first filter fails, we try the second, and passing either filter is sufficient to pass a data item. An example is the ("burro" or "donkey") subquery in section 1.

Optimization of disjunctions is precisely analogous to (the "dual" of) optimization of conjunctions. The following theorem generalizes Theorem 2 of [15] beyond independent filters. Note one peculiarity of disjunctions is that entailing filters must precede their entailed filters to make sense, since if the success of filter A implies the success of filter B, then failure of filter B implies the failure of filter A, and it is failure that causes us to continue to the next filter in sequence in a disjunction.

Theorem 5.1: The problem of optimally ordering a disjunctive sequence of filters is equivalent to optimally ordering a conjunctive sequence in which the costs are the same, probabilities are mapped to their inverses, and entailment relationships are reversed. Proof: If the filters are applied sequentially, everything that fails the first filter is applied to the second filter, and everything that fails both filters is applied to the third filter, and so on. The final answer is just the appending of results from all filters, since this union is disjoint. Hence the cost formula is |c sub 1 + c sub 2 p (" " f sub 1) + c sub 3 p (" " f sub 1 andsign " " f sub 2) + ... |, identical to that

for conjunctive sequences except with filter-failure events rather than filter-success events. But the probability of filter success is just the inverse of the probability of filter failure, and the inverse function |f (x) = 1 - x| is monotonic with the same domain and range. When the inverse is taken, all entailment relationships |A -> B| can be transformed into relationships in the reverse direction, |" " B -> " " A|. So the problem of finding an optimal disjunctive sequence has an exact "dual" problem of finding the optimal conjunctive sequence for the inverse of the original probabilities with reversed entailments. The solution to the latter found by the abovementioned methods then maps to the solution for the former. QED.

Note that this also provides criteria for deletion of redundant disjunctive filters, and implies that concurrent execution of different filters in disjunctions is also not advantageous.

The distributive laws

If a boolean expression referring to some filters (a "general boolean filter") includes both conjunctions and disjunctions, will factoring it (using the distributive laws) improve execution time? Surprisingly, the answer is an unequivocal "yes."

Theorem 5.2: With three arbitrary nontrivial filters, the boolean filter |f sub 1 andsign (f sub 2 orsign f sub 3)| is faster than the boolean filter |(f sub 1 andsign f sub 2) orsign (f sub 1 andsign f sub 3)|. Proof: Let |u| represent all events before |f sub 1| is evaluated. The first method is better than the second if:

```
c sub 1   +   c sub 2 p ( f sub 1 | u )   +   c sub 3 p (
f sub 1 andsign " " f sub 2 | u )   <   [ c sub 1   +   c sub 2 p ( f
sub 1 | u ) ]   +   [ c sub 1 p ( " " f sub 1 orsign " " f sub 2 | u )
  +   c sub 3 p ( f sub 1 andsign " " f sub 2 | u ) ]
```

which simplifies to |0 < c sub 1 p (" " f sub 1 orsign " " f sub 2 | u)|, which is always true for nontrivial filters. QED.

Hence an additional local optimality condition for a general boolean filter involving both conjunctions and disjunctions is that all possible factorings be made for conjunctions over disjunctions. A similar result holds for disjunctions over conjunctions.

Theorem 5.3: The boolean filter |f sub 1 orsign (f sub 2 andsign f sub 3)| is faster than the boolean filter |(f sub 1 orsign f sub 2) andsign (f sub 1 orsign f sub 3)| . Proof: The first method is better than the second if:

```
c sub 1   +   c sub 2 p ( " " f sub 1 | u )   +   c sub 3 p ( " " f sub
1 andsign f sub 2 | u )   <
[ c sub 1   +   c sub 2 p ( " " f sub 1 | u ) ]   +
[ c sub 1 p ( f sub 1 orsign f sub 2 | u )   +   c sub 3 p ( " " f
sub 1 andsign f sub 2 | u ) ]
```

And all terms cancel except for the third on the right side, |c sub 1 p (f sub 1 orsign f sub 2 | u)|. QED.

Note that Theorems 5.2 and 5.3 do not require probabilistic independence. And the |f sub i| terms can be composite (or boolean filters themselves), so the result applies to the distribution of conjunction over more than two disjunctions, and vice versa. But these results are only local optimality conditions, because some boolean expressions can be factored in more than one way. For instance, there are two local optima for:

```
( a andsign b )   orsign   ( a andsign c )   orsign   ( b andsign c )
   =    ( a andsign ( b orsign c ) )   orsign   ( b andsign c )
   =    ( a andsign b )   orsign   ( ( a orsign b ) andsign c )
```

However, this problem can usually be ignored, since conjunctions are the most common way to conjoin filters, and it is rare that a filter must appear twice in any locally-optimal execution plan as above.

Redundancy elimination in boolean expressions

A special case of the distributive law is an "absorption" law:

```
f sub 1 orsign ( f sub 1 andsign f sub 2 )   =
( f sub 1 andsign true ) orsign ( f sub 1 andsign f sub 2 )   =
f sub 1 andsign ( true orsign f sub 2 )   =   f sub 1
```

The final expression must execute faster than the original expression because it is a subexpression. The other useful absorption law is |f sub 1 andsign (f sub 1 orsign f sub 2) = f sub 1|.

In general, all such redundancy-elimination laws of logic can be fruitfully applied to general boolean filters. They include:

```
f andsign f = f ,   f orsign f = f ,   f andsign true = f ,   f andsign false
= false ,   f orsign true = true ,   f orsign false = f
```

All these permit replacement by an expression requiring less work to evaluate.

Negations

Negation operators will complete a boolean algebra of filter expressions. Negations can be thought of as set differences on the results of filters previously passed, if any, otherwise on the full database. Double negations can be eliminated.

We will assume that the negation of a noncomposite filter has the same execution cost as the unnegated filter on the same set. This is true for signature tables and other filters wherein a similar calculation is performed upon every input data item, and the result used to decide if the item passes the test; negation then just means switching the sense of the final comparison. More complex filters that do not fulfill this restriction can often be decomposed into boolean combinations of subfilters that do. Under this assumption, we can prove that negations in a boolean filter expression should be pushed as far as possible inside expressions, so that they all apply to single filters and thus can be evaluated at no cost penalty.

Theorem 5.4: Consider an equivalence class of boolean expressions such that any member of the class can be transformed into any other member by some sequence of applications of DeMorgan's Laws. Then if this expression is interpreted as referring to information filters, the globally optimal member of the class is that in which every negation is of a noncomposite (simple) filter. Proof: There must be only one such expression, because DeMorgan's Laws that move negations inward apply independently to each negation sign. Any other expression in the equivalence class must be derivable by a series of applications of the reverse laws.

First consider |" " f sub 1 andsign " " f sub 2| versus |" " (f sub 1 orsign f sub 2)|. If |u| represents the "context" of the subexpressions, the logical conditions in effect when these subexpressions are reached during execution, and if |f sub 1| and |f sub 2| are simple filters, the first expression costs |c sub 1 + p (" " f sub 1 | u) c sub 2|, and the second costs |c sub 1 + p (" " f sub 1 | u) c sub 2 + c sub n|, where |c sub n| is the "negation cost", the cost per data item of checking which items in a set do *not* belong to |u|. Hence with noncomposite filters, the first form is always better because all the terms are the same except for the added negation-cost term in the second expression. Second, consider |" " f sub 1 orsign " " f sub 2| versus |" " (f sub 1 andsign f sub 2)|. The first

expression costs |c sub 1 + p (f sub 1 | u) c sub 2|, and the second costs |c sub 1 + p (f sub 1 | u) c sub 2 + c sub n|. Again, the second cost is worse for noncomposite filters.

If |f sub 1| and |f sub 2| are composite in the above filter expressions, the transformation from the first form to the second might be thought to decrease cost if a double negation could cancel. However, the cost calculation for the second form of each pair always adds at least one |c sub n| term. Hence its cost is always worse than that of an expression in the same equivalence class with no |c sub n| terms in its cost, the expression with negations pushed all the way inward, since the |c sub n| terms are the only terms that can be affected by DeMorgan's Laws. QED.

Besides DeMorgan's, a few other laws of logic that can help simplify an expression. |f sub 1 andsign " " f sub 1 = false | and |f sub 1 orsign " " f sub 1 = true | are desirable substitutions. The "negative absorption" laws |f sub 1 andsign (" " f sub 1 orsign f sub 2) = f sub 1 andsign f sub 2 | and |f sub 1 orsign (" " f sub 1 andsign f sub 2) = f sub 1 orsign f sub 2| also eliminate useless work when the inner expression is evaluated first, and do no harm otherwise.

As an example of these and several other results of section 5, consider the query:

("burro" and not ("side view" or "desert")) or ("burro" and "desert")

which will be improved by rephrasing as:

("burro" and not "side view" and not "desert") or ("burro" and "desert")

which will be improved by rephrasing as:

"burro" and ((not "side view" and not "desert") or "desert")

and cannot be worsened by rephrasing as:

"burro" and (not "side view" or "desert")

Now costs and probabilities could be introduced to find best orders for the conjunction and disjunction.

A summary table

Fig. 12 summarizes the optimization techniques of this paper. We have covered the laws of boolean logic listed in [16] and so have covered everything useful for the propositional calculus. The middle column shows that the first four classes of logical equivalences do require some cost and probability analysis; but the methods of section 2 will do this, and they are not computationally expensive. The rightmost column shows that three of the seven classes of equivalences are possible difficulties for a "greedy" algorithm which sequentially applies the best equivalence until it reaches a local optimum. These three are what make the general problem difficult and probably exponential in complexity. But results of section 4 suggest that a polynomial-time greedy algorithm can get the right answer most of the time.

If this is insufficient assurance of optimality of a query execution plan, standard optimization techniques [13] can search the space of possible expressions, guided by the constraints we have formulated. The technique of simulated annealing may be especially helpful here because of its recent success on similar combinatorial problems.

Experiments with a natural-language processing-filter application

We now discuss a specific filtering application to which we have applied our theory. This application illustrates a number of subtleties in the use of filters. It also is valuable in its own right, as one of the easiest ways for users to access multimedia databases. The idea is provide information retrieval of multimedia data with natural-language (in our case, English) questions as input. Examples of this approach are [14] and [25] and the more complicated ideas reviewed in [26]. Natural-language processing poses a good challenge for filtering ideas because it can require much time, yet it is not so slow as to fail to increase user satisfaction when its efficiency improves.

We wish to improve our earlier MARIE-1 system [24]. It takes as input a English noun phrase representing a query, and returns as output the multimedia data items that match the meaning (as opposed to the words) of the query, doing most processing with the pointers to those data items and not the items themselves. The domain of MARIE-1 is captioned photographs in the Photo Lab of the U.S. Navy air test facility NAWC-WD, China Lake, California. The conceptual units of the improved MARIE-1 will be:

1. *Coarse-grain matcher (C)*: a keyword match of the set of nouns in the English query input to caption nouns, using index files [24]. It returns a set of pointers to media data items. It uses a type hierarchy to permit type-subtype matches, an important feature for helpful information retrieval, as discussed in [6] and [29].

2. *Parser (P)*: a natural-language understanding system that parses English query input and creates a *meaning list*, its logical form [24]. We assume input and captions exhibit "conjunctive semantics" [1] where the meaning of the whole is the conjunction of a set of logical expressions that define the meaning of the parts, a usually reasonable assumption for captions because of their concrete subjects.

3. *Registration-data tester (R)*: a formatted-condition processor, like those in database query languages, that returns pointers to data items matching registration-data (formatted non-caption) conditions in the query input. Registration data at NAWC-WD includes date, location, photographer, type of film, and security classification. This tester was implemented for this paper using a main-memory database.

4. *Picture-type matcher (T)*: an identifier of the possible broad classes to which a media datum or a query can belong (like "test" or "historical" or "public relations" for photographs), which then rules out media data whose classes are incompatible with the query classes [23].

5. *Fine-grain matcher (F)*: a graph matcher that checks whether the query input graph (representing the query meaning list) is isomorphic to some part of some caption graph (representing the caption's meaning list) [24]. Like the coarse-grain matcher, this needs a type hierarchy, and it also needs a part-whole hierarchy. It helps to separate this from the coarse-grain match, as did [7, 21], since fine-grain requires combinatorial analysis and can be much slower.

6. *Shape analysis (S)*: a picture-processing routine that partitions the picture into regions of similar color and texture, computes statistics on all of them, and selects possible identifications for them from a few general categories ("aircraft", "sky", "person", etc.). None of the test queries we obtained from users could exploit such a filter (since users knew the existing keyphrase system could not handle such information), so we omitted it from our final test. An example usage would be for "size greater than 20% of the picture" in the section 1 example.

Fig. 13 shows the dependencies between the conceptual units. Each of these can be a separate process on a separate processor; input and output queues can enable asynchronous communication. Items 1, 3, 4, 5, and 6 are conjunctive information filters as we defined them in section 2, and substantial filters at that; and item 2 imposes a minimum start-time constraint on 4 and 5. An additional final filter could be a human user who accepts or rejects what the computer eventually supplies. We could also add separate filters for additional textual, audio, and video aspects of a datum, if these could be analyzed separately.

Mathematical analysis of the four MARIE filters

We can consider all possible orderings of the filter processes excluding filter S. Entailments can be summarized:

--C-R: independent (they examine different sorts of data)
--C-T: approximately independent (two different kinds of reasoning)
--C-F: first entailed by second
--R-T: independent (they examine different sorts of data)
--R-F: independent
--T-F: first entailed by second

Then the only possible filter sequences obeying dependencies are:

4-filter: CRTF, CTRF, RCTF, RTCF, TCRF, TRCF, CTFR, TCFR
3-filter: RTF, CRF, TRF, RCF, TFR, CFR
2-filter: RF, FR

We randomly chose 230 captions from the NAWC-WD Photo Lab database. We used 44 test queries, 42 supplied by the Photo Lab personnel as typical of the queries they receive everyday, and two longer queries from [23]. We obtained average CPU times per data item (in seconds), and average success probabilities, in experiments with an implementation in Quintus Prolog for Unix on a Sun SparcStation. These measurements, plus the ratios |r sub i = c sub i / (1 - p sub i)|, were:

|c sub C = 0.0102 , p sub C = 0.0305 , r sub C = 0.0105|
|c sub R = 0.000602 , p sub R = 0.958 , r sub R = 0.0144|
|c sub T = 0.000236 , p sub T = 0.749 , r sub T = 0.000939|
|c sub F = 3.11 , p sub F|C&T = 0.421 "(F conditional probability given C and T)," r sub F = 5.37|

Note how a redundant filter can still be highly useful: Coarse-grain rules out an average of 97% of the database in very little time.

If we could ignore the parser (as when for certain common queries, meaning lists are stored in advance), the interchange-entailment sort of these four filters would be TCRF (picture-type matcher, coarse-grain matcher, registration-data tester, and fine-grain matcher). This order satisfies the dependencies. It would also satisfy deletion optimality since |0.000939 < 0.0102| for deletion of T and |0.105 < 0.00602 + (0.958 * 3.11)| for deletion of C. C would be strongly-deletion-optimal since it is not followed by an entailed (deletable) filter; T would be strongly-deletion-optimal because |0.000236 < 0.0102|. F would satisfy strong interchange-optimality because R cannot be deleted, and |0.0144 < 0.421|. Hence TCRF would be globally optimal.

But the parser (P) imposes an minimum time for T and F from the start of processing. P is going to be on the critical path for the optimal execution plan, because even if C and R are done sequentially while P is executing, |0.0102 + (0.0305 * 0.00602) = 0.0104| is the expected time for the sequence C-R, and we obtained an average

of |c sub P = 3.76 / 230 = 0.0163| seconds for parse CPU time per data item for the average query. Hence P and F will be on the critical path (F cannot be deleted), and must occur in that order; if T occurs, it must be between P and F because of the dependencies. T must occur because it is strongly-deletion-optimal, since |0.000236 < 3.11|. Hence the critical path is parser-T-F.

Just for comparison, shape-analysis filter S works on 100 by 100 pixel reductions of the photographs, and took about 2000 seconds per picture, much slower than the other filters. So |r sub S > 2000| and it should definitely be applied after the other filters when it is used.

To experiment with parallel processing, we used the Quintus Prolog communications package TCP using 1, 2, 3, and 4 processors. The parser cannot be decomposed into parallel tasks as currently implemented, and parallelism is of only occasional advantage to C and R because they are not on the critical path on the average, even if taken sequentially. So the best we can do is to execute the sequence C-R for all data items on a processor run in parallel with P. That leaves T and F for possible data-partition parallelism. In separate experiments, we observed no statistically significant effect of the number of processors on overhead cost (that is, the |k sub 1| of section 3.1 was negligible). So we fit cost to the formula |k sub 0 + (c sub F / N)| for the cost of distributing fine-grain over |N| processors, and found |k sub 0 = 0.4| seconds as the average overhead per data item. Hence T and F should be allocated the maximum number of available processors each, and all of T's filtering must precede all of F's filtering, following Theorem 3.2. Fig. 14 summarizes the optimal execution plan.

To confirm the preceding analysis, we conducted further tests. Among other things, we measured real time to the nearest second for execution of the four filter sequences CF, CTF, F, and TF on a Sun SparcStation. We measured real time to be sure to include all the factors that affect execution time, but the workstation used a fileserver that runs background processes and serves other users, so our measurements were not precise. In 42 queries of the previously supplied 44 (two of which showed new bugs and were excluded), the observed ratio of real execution time for F to TF was 1.18 with a standard deviation of 0.43, versus a theoretical ratio of 1.33; and the ratio of F to CF was 22.1 with a standard deviation of 17.3, versus a theoretical ratio of 29.7. The observed ratio of real execution time for CF to CTF was 2.20 with a standard deviation of 1.50, versus a theoretical ratio by our above methods of 1.33; the data was in the form of small integers averaging about 20, so this measurement was more crude. In these same experiments, F was never faster than CF, and TF was never faster than CTF, as predicted by theory; and F was faster than TF as predicted in only 9 out of 42 cases, and only slightly faster in each of the 9. These results are adequate confirmation of our theory considering that our method of parameter estimation, by averaging over all queries, biases performance toward the few queries with many answers.

Scaling up the database

The MARIE-1 system was just a prototype implementation that handled a random sample of 1/166 of the entire NAWC-WD database, which at the time of the sample was 36,000 captioned data items. We can use the preceding analysis to predict the optimal execution plan when MARIE is applied to the full database.

The time to do fine-grain matching and picture-type matching for a single data item should increase by 166 since each match is independent and there are no economies of scale. Registration-data testing will be close to 166 times slower because it is best implemented with indexes, and the indexes will be 166 times longer on the average. Coarse-grain matching will be dominated by the intersection of lists 166 times longer on the average, so it will also be close to 166 times slower. Only the parser will remain nearly the same speed, since the grammar it handles is nearly a complete grammar for the remaining captions, and most of the parse time is in fetch from a hashed lexicon, backtracking among grammar choices, and reasoning about lexicon information for words, all activities not affected by the size of the database.

That means the effective time cost for P is reduced to |3.76 / 36000 = 0.000104| per data item. This still rules out the sequence TCRF since T must wait for P to conclude, but still allows the sequences CRTF, CTRF, RCTF, RTCF, CTFR, RTF, CRF, RCF, CFR, RF, and FR. Deletion optimality is not affected by the inclusion of P, so we need only consider the four-filter sequences CRTF, CTRF, RCTF, RTCF, and CTFR. The optimal solution when a single inequality constraint is imposed upon a problem is one in which the inequality constraint is "active" or at the border of infeasibility, which means T must be second in the sequence, leaving only CTRF, RTCF, and CTFR as possibilities. But in the second of those R precedes C, and in the third of those F precedes R, both of which are inconsistent with interchange optimality. Hence sequence CTRF is the optimal one, with P in parallel with C. As before, there is no benefit to putting any two filters in parallel, but there is an advantage in data-partition parallelism on each filter. So the optimal execution plan with N available filters is to put 1 processor on P in parallel with N-1 processors on C, then N processors on the sequence TRF on different random partitions of the database.

Other modifications of MARIE

Our analysis also permits straightforward analysis of several interesting hypothetical modifications. If the parser finds natural-language input to be ambiguous, alternative meaning lists can be generated. Then the T and F processors can test a disjunction of the alternatives, using the methods of section 5.

Another idea is to split the coarse-grain filter into separate filters for each noun of the query. The current implementation uses a single heap structure for all input nouns, but a simpler implementation would load and intersect index files, finding captions belonging to each of a set of index files (assuming an exact match to the query is desired). Then large index files have both a high cost and a high probability of success, hence a high ratio |c / (1 - p)|. Hence if we partition the coarse-grain matching into separate filters for each noun, the filters should be sorting by increasing frequency of noun occurrence. This may mean that other filters like R and T can now be interleaved with coarse-grain filters, and placed before filters for unhelpful high-frequency nouns (like the frequent words "view" and "test" at NAWC-WD).

Yet another idea is to treat the user as another filter U, as was suggested. The user will examine data items supplied, and will accept some of them. We can assume that any user knows better what they want than any automatic filter, so this "user filter" U entails all the others discussed; but to maintain a tree structure for entailments, U must only directly entail the filter that must otherwise go last, the shape-analysis filter S. The deletion of S in the presence of U is desirable if |2000 / N > c sub U| as a sufficient condition, N the number of processors, and this seems undeniable as long as time is the criterion and there are fewer than 100 processors, because most users can assess a picture in 10 seconds. (If other criteria were included in the cost like bother to the user, this threshold would decrease.) So S should be deleted if U is present. Then F will be inherit entailment by U, and F should be deleted if time is the only criterion and the user averages less than |5.37 / N| seconds to assess a picture.

Further capabilities with a mixed query language

Beyond the capabilities just described, we have implemented for MARIE-1 an enhanced query capability in a SQL-like format, allowing arbitrary nesting of boolean expressions, including possibly multiple natural-language strings and multiple registration-data restrictions. The methods of sections 2, 3, and 5 can be applied to these enhanced queries. Our query language adds a comparator "MATCHES" to SQL, to initiate natural-language processing and semantic matching. The query language does not handle joins, however, since we believe that good captions and a good type hierarchy can eliminate most need for them in multimedia databases. The availability of such a "mixed" query language with both conventional SQL and natural-language-descriptor

features means that the natural-language processing does not need to handle complicated scoping rules for quantifiers like "not", "or", and "all", which can be very tricky to analyze in English, since the user can express such distinctions with the formal part of the query language. Programmers can use our modified SQL directly, but we also provide a graphical interface for naive users that permits structured query formulation.

Conclusions

We have explored a new approach to information retrieval, the concept of information filtering. Our approach has focused on the system aspects of filtering rather than the details of the filters, and our work should complement the results on filter design provided by classic information-retrieval methods and work on signature-based retrieval. Our approach has been mostly analytical, providing local optimality criteria for filter execution plans. Thus it contrasts with work on query optimization for database systems, for which the search space is so difficult to analyze that methods must be either exhaustive or heuristic; using our local optimality criteria is several degrees better than the "cheapest-first" heuristic often used there. Information filtering thus appears to be a special case of general database retrieval that has special exploitable properties for improving efficiency. The methods we have proposed will be particularly useful for the design of multimedia information-retrieval systems, for which conceptually distinct filters can easily be derived.

References

[1] Allen, J. *Natural language understanding*. Menlo Park, CA: Benjamin Cummings, 1987.

[2] Belkin, N. J. and Croft, W. B. Information filtering and information retrieval: two sides of the same coin? *Communications of the ACM, 35*, 12 (December 1992), 29-38.

[3] Bertino, E., Rabitti, F., and Gibbs, S. Query processing in a multimedia document system. *ACM Transactions on Office Information Systems, 6*, 1 (January 1988), 1-41.

[4] Chang, J. W., Hyuk, J.C., Sang, H. O., and Lee, Y. J. Hybrid access method: an extended two-level signature file approach. International Conference on Multimedia Information Systems, ACM, Singapore (1991), 51-62.

[5] Chang, S. K., Yan, C. W., Dimitroff, D. C., Arndt, T. An intelligent image database system. *IEEE Transactions on Software Engineering, 14*, 5 (May 1988), 681-688.

[6] Chen, H., Lunch, K. J., Basu, K., and Ng, D. T. Generating, integrating, and activating thesauri for concept-based document retrieval. *IEEE Expert, 8*, 2 (April 1993), 25-34.

[7] Cohen, P. R. and Kjeldsen, R. Information retrieval by constrained spreading activation in semantic networks. *Information Processing and Management, 23*, 4 (1987), 255-268.

[8] Constantopoulos, P., Drakopoulos, J., and Yeorgaroudakis, Y. Retrieval of multimedia documents by pictorial content: a prototype system. International Conference on Multimedia Information Systems, ACM, Singapore (1991), 35-48.

[9] El-Rewini, H., Lewis, T., and Ali, H. H. *Task scheduling in parallel and distributed systems*. Englewood Cliffs, NJ: Prentice-Hall, 1994.

[10] Faloutsos, C. Signature-based text retrieval methods: a survey. *Database Engineering*, March 1990, 27-34.

[11] Faloutsos, C. and Christodoulakis, S. Description and performance analysis of signature file methods for office files. *ACM Transactions on Office Automation Systems, 5*, 3 (July 1987), 237-257.

[12] Gibbs, S., Tsichritzis, D., Fitas, A., Konstantas, D., and Yeorgoroudakis, Y. (1987). Muse: A multimedia filing system. *IEEE Software, 4*, (2), 4-15.

[13] Gill, P. E., Murray, W., and Wright, M. H. *Practical optimization*. London: Academic Press, 1981.

[14] Grosz, B., Appelt, D., Martin, P. and Pereira, F. TEAM: An experiment in the design of transportable natural language interfaces. *Artificial Intelligence, 32* (1987), 173-243.

[15] Hanani, M. Z. An optimal evaluation of boolean expressions in an online query system. *Communications of the ACM, 20*, 5 (May 1977), 344-347.

[16] Jarke, M. and Koch, J. Query optimization in database systems. *Computing Surveys, 16*, 2 (June 1984), 117-157.

[17] Krovez, R. and Croft, W. B. Lexical ambiguity and information retrieval. *ACM Transactions on Information Systems, 10,* 2 (April 1992), 115-141.

[18] Lee, Y.-H. and Krishna, C. Optimal scheduling of signature analysis for VLSI testing. *IEEE Transactions on Computers, 40*, 3 (March 1991), 336-340.

[19] Mine, H. and Osaki, S. *Markovian decision processes*. New York: American Elsevier, 1970.

[20] Pattipatti, K. and Dontamsetty, M. On a generalized test sequencing problem. *IEEE Transactions on Systems, Man, and Cybernetics, 22*, 2 (March/April 1992), 392-396.

[21] Rau, L. Knowledge organization and access in a conceptual information system. *Information Processing and Management, 23*, 4 (1987), 269-284.

[22] Roussopoulous, N., Faloutsos, C., and Sellis, T. An efficient pictorial database system for PSQL. *IEEE Transactions on Software Engineering, 14,* 5 (May 1988), 639-650.

[23] N. C. Rowe, Inferring depictions in natural-language captions for efficient access to picture data. *Information Processing and Management, 30*, 3 (1994), 379-388.

[24] Rowe, N. and Guglielmo, E. Exploiting captions in retrieval of multimedia data. *Information Processing and Management, 29*, 4 (1993), 453-461.

[25] Sembok, T. and van Rijsbergen, C. SILOL: A simple logical-linguistic document retrieval system. *Information Processing and Management, 26*, 1 (1990), 111-134.

[26] Shekhar, S., Srivastava, J., and Dutta, S. A formal model of trade-off between optimization and execution costs in semantic query optimization. *Data and Knowledge Engineering, 8* (1992), 131-151.

[27] Smeaton, A. F. Progress in the application of natural language processing to information retrieval tasks. *The Computer Journal, 35*, 3 (1992), 268-278.

[28] Smith, D. and Genesereth, M. Ordering conjunctive queries. *Artificial Intelligence, 26,* (1985), 171-215.

[29] Smith, P., Shute, S., Galdes, D., and Chignell, M. Knowledge-based search tactics for an intelligent intermediary system. *ACM Transactions on Information Systems, 7,* 3 (July 1989), 246-270.

[30] Stanfill, C. and Kahle, B. Parallel free-text search on the Connection Machine system. *Communications of the Assocation for Computing Machinery, 29*, 12 (December 1986), 1229-1239.

```
filter(2,2.18817,[0.0621229,dependent,1]).
filter(3,8.6706,[0.0347425,dependent,2]).
filter(4,8.63665,[0.0574234,independent]).


Cost-prob ratios: [4.78325,2.33311,8.98268,9.16281]


[1,2,4,3]: 4.81338
[2,4,3]: 2.75564 (local optimum)
[3,4]: 8.97066 (local optimum)
[1,4,3]: 5.68422


The number of local optima: 2
Heuristic optimum: [2,4,3]
```

Figure 1: Example output of the conjunctive-sequence tester

---

no. of filters	*prob. entailed*	*prob. entailing*	*heuristic size of space*	*number of optima*	*cost ratio*
3	*0.2*	*0.2*	*0.0554518(0.0188046)*	*0.0138629(0.00970406)*	*0.0(0.0)*
4	*0.2*	*0.2*	*0.152492(0.0303405)*	*0.00693147(0.00689673)*	*0.0(0.0)*
5	*0.2*	*0.2*	*0.263396(0.0446852)*	*0.038712(0.0172595)*	*0.0(0.0)*
6	*0.2*	*0.2*	*0.505997(0.0641714)*	*0.0762462(0.0216879)*	*0.0(0.0)*
7	*0.2*	*0.2*	*0.526792(0.0651116)*	*0.112081(0.0272095)*	*0.0(0.0)*
8	*0.2*	*0.2*	*0.672353(0.0749465)*	*0.114958(0.0266312)*	*0.0(0.0)*
9	*0.2*	*0.2*	*0.977338(0.0930445)*	*0.156547(0.0322006)*	*0.0(0.0)*
10	*0.2*	*0.2*	*1.21301(0.113208)*	*0.168711(0.0340448)*	*0.0(0.0)*
11	*0.2*	*0.2*	*1.2338(0.116638)*	*0.238026(0.039367)*	*0.0(0.0)*
12	*0.2*	*0.2*	*1.45561(0.133149)*	*0.235557(0.0417515)*	*0.0145623(0.0144893)*
13	*0.2*	*0.2*	*2.07251(0.145889)*	*0.343708(0.0472216)*	*0.0(0.0)*
14	*0.2*	*0.2*	*2.25273(0.147813)*	*0.35064(0.0516756)*	*0.0(0.0)*
15	*0.2*	*0.2*	*2.12796(0.161744)*	*0.327977(0.054881)*	*0.0125881(0.00800236)*
16	*0.2*	*0.2*	*2.74486(0.170891)*	*0.471194(0.0614881)*	*0.0151046(0.0142948)*
17	*0.2*	*0.2*	*3.04985(0.179684)*	*0.478125(0.0579669)*	*0.0008991(0.0006283)*
18	*0.2*	*0.2*	*3.39642(0.185603)*	*0.622168(0.0740848)*	*0.0008681(0.0008096)*
19	*0.2*	*0.2*	*4.07571(0.212578)*	*0.832629(0.0828184)*	*0.0521339(0.0289764)*
20	*0.2*	*0.2*	*4.19354(0.199061)*	*0.695475(0.0637804)*	*0.0360905(0.0264117)*
3	*0.2*	*0.8*	*0.277259(0.0449211)*	*0.00693147(0.00689673)*	*0.0(0.0)*
4	*0.2*	*0.8*	*0.547586(0.0557068)*	*0.0207944(0.0118242)*	*0.0(0.0)*
5	*0.2*	*0.8*	*0.998132(0.065259)*	*0.103972(0.0247503)*	*0.0(0.0)*
6	*0.2*	*0.8*	*1.26846(0.0832094)*	*0.128821(0.0294552)*	*0.0(0.0)*
7	*0.2*	*0.8*	*1.85763(0.0887012)*	*0.253066(0.0391569)*	*0.0120575(0.00854089)*
8	*0.2*	*0.8*	*2.38443(0.10481)*	*0.360936(0.0487895)*	*0.0130564(0.012991)*
9	*0.2*	*0.8*	*2.80725(0.0986974)*	*0.46442(0.0569815)*	*0.0332153(0.0252187)*
10	*0.2*	*0.8*	*3.16768(0.10009)*	*0.550636(0.056419)*	*0.0347772(0.0245527)*

11	0.2	0.8	4.08957(0.108937)	0.619519(0.0599677)	0.0317728(0.0316135)
12	0.2	0.8	4.42921(0.113004)	0.708135(0.0682391)	0.018403(0.016857)
13	0.2	0.8	5.15702(0.114451)	0.664393(0.068758)	0.0346816(0.0187861)
14	0.2	0.8	5.85709(0.100626)	0.78386(0.0816895)	0.0176791(0.0119071)
15	0.2	0.8	6.40468(0.1131)	0.89792(0.0758427)	0.0402969(0.0223391)

Figure 2, page 1

---

no. of filters	prob. entailed	prob. entailing	heuristic size of space	number of optima	cost ratio
3	0.8	0.2	0.270327(0.448086)	0.0277259(0.0135829)	0.0(0.0)
4	0.8	0.2	0.60997(0.681121)	0.0277259(0.0135829)	0.0(0.0)
5	0.8	0.2	0.89416(0.971866)	0.0415888(0.0164613)	0.0(0.0)
6	0.8	0.2	1.44175(0.119125)	0.0855333(0.0264176)	0.0(0.0)
7	0.8	0.2	1.6081(0.144361)	0.157725(0.0355865)	0.0(0.0)
8	0.8	0.2	2.18341(0.170456)	0.122422(0.0320054)	0.0135923(0.0132327)
9	0.8	0.2	3.02212(0.174673)	0.156026(0.0367928)	0.00639268(0.00636064)
10	0.8	0.2	3.37563(0.190245)	0.194092(0.0383726)	0.00763135(0.00753294)
11	0.8	0.2	3.81924(0.219739)	0.262072(0.0491324)	0.0782128(0.0489747)
12	0.8	0.2	4.92134(0.222132)	0.251086(0.0458398)	0.049436(0.0312112)
13	0.8	0.2	5.55211(0.233519)	0.229114(0.0449624)	0.0930229(0.039692)
14	0.8	0.2	5.46893(0.269864)	0.326485(0.0540142)	0.0569272(0.0359603)
15	0.8	0.2	6.01652(0.261727)	0.362034(0.0564617)	0.0817849(0.0308793)
3	0.8	0.8	1.04665(0.0432815)	0.0207944(0.0118242)	0.0(0.0)
4	0.8	0.8	1.6081(0.0562091)	0.0762462(0.0216879)	0.0(0.0)
5	0.8	0.8	2.39829(0.0549469)	0.0733694(0.0223444)	0.0(0.0)
6	0.8	0.8	3.02212(0.0567535)	0.0872323(0.0239395)	0.0219433(0.0218333)
7	0.8	0.8	3.80538(0.0567323)	0.15367(0.0318464)	0.0616014(0.02526)
8	0.8	0.8	4.47773(0.0513303)	0.180875(0.0390725)	0.0680312(0.0468534)
9	0.8	0.8	5.1986(0.0541365)	0.162188(0.0372266)	0.0712856(0.03593)
10	0.8	0.8	5.8363(0.0538518)	0.193968(0.0404172)	0.0648062(0.0241476)
11	0.8	0.8	6.63342(0.0482771)	0.242613(0.0448573)	0.118454(0.0395468)
12	0.8	0.8	7.30577(0.0549469)	0.222182(0.0436092)	0.164913(0.052674)

Figure 2, page 2

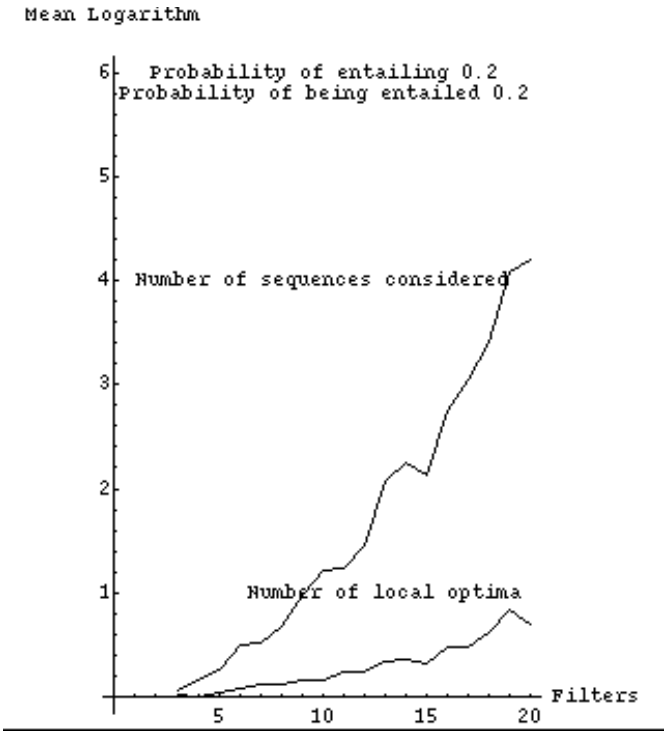

Figure 3: Problem difficulty: Case 1.

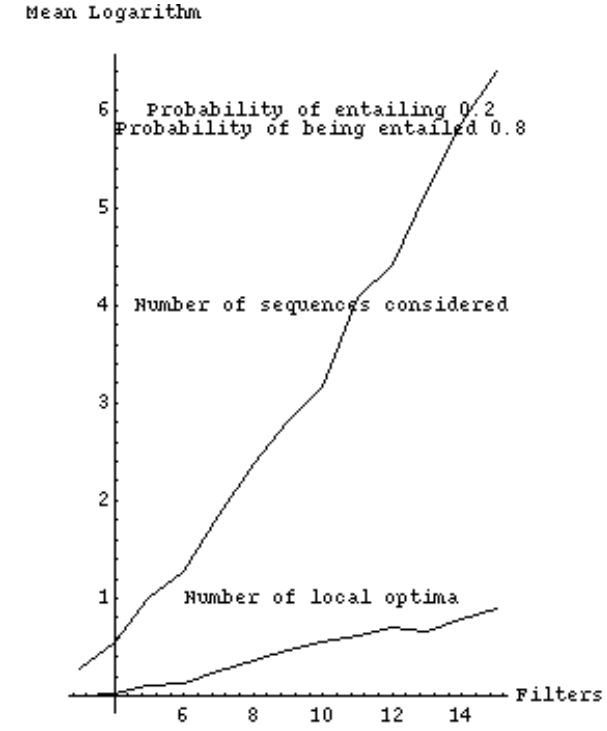

Figure 4: Problem difficulty: Case 2.

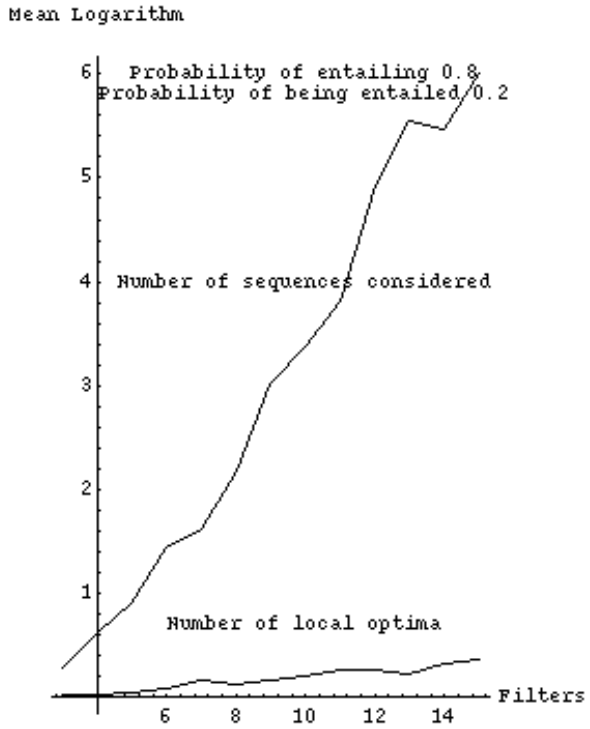

Figure 5: Problem difficulty: Case 3.

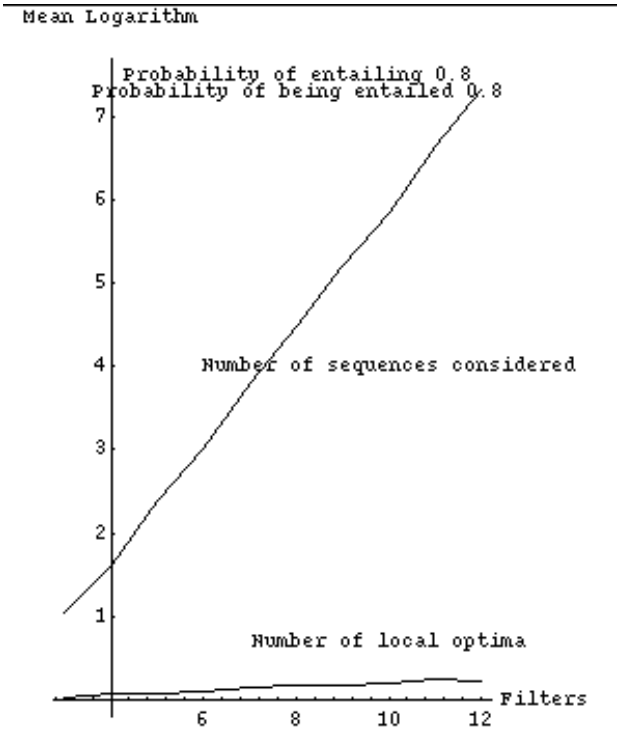

Figure 6: Problem difficulty: Case 4.

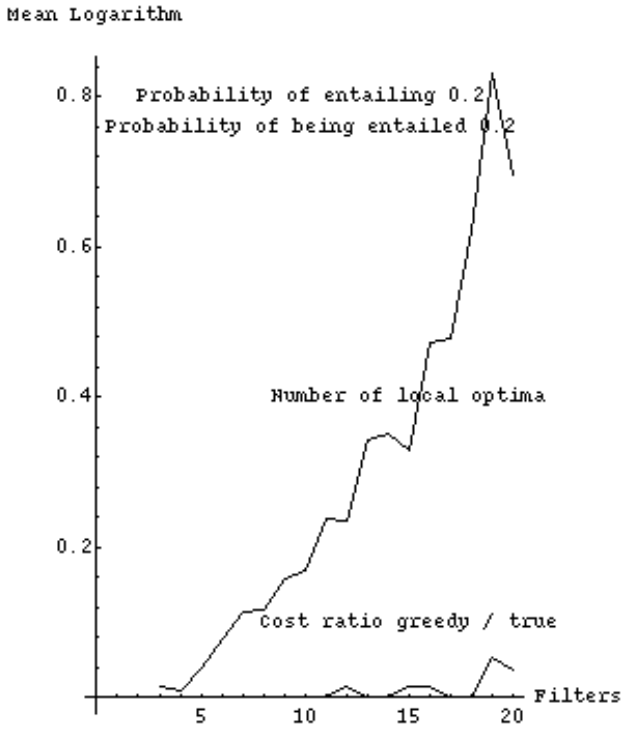

Figure 7: Answer quality: Case 1.

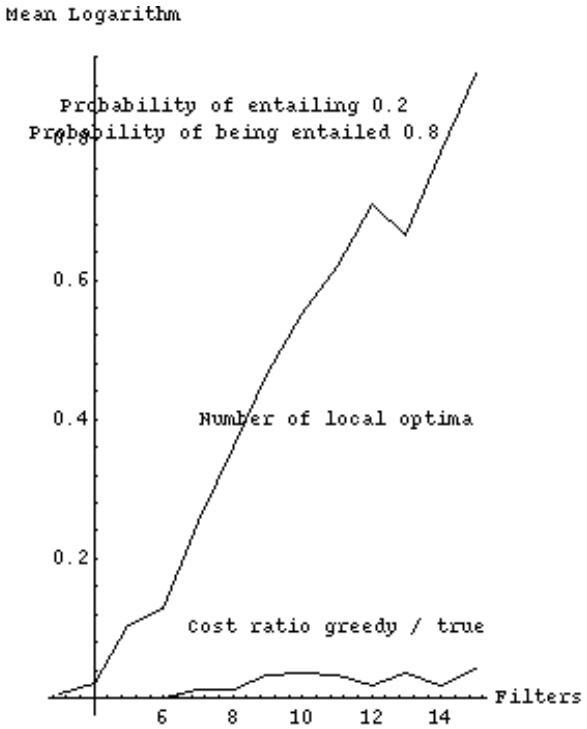

Figure 8: Answer quality: Case 2.

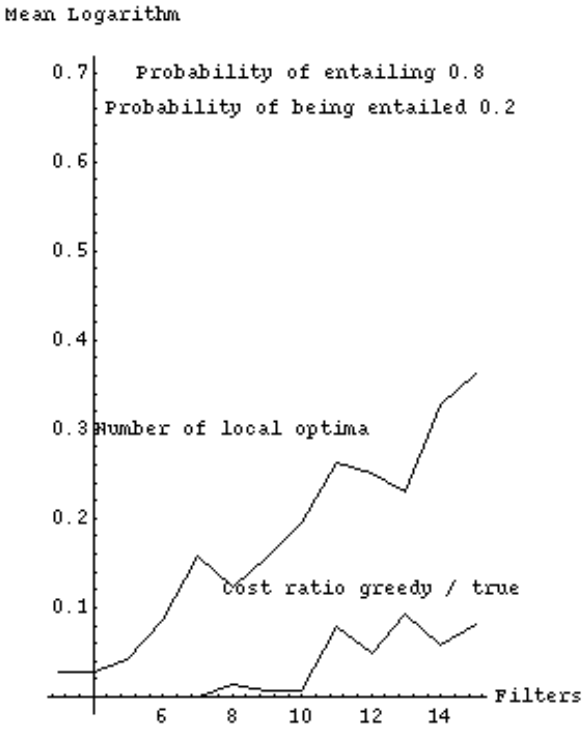

Figure 9: Answer quality: Case 3.

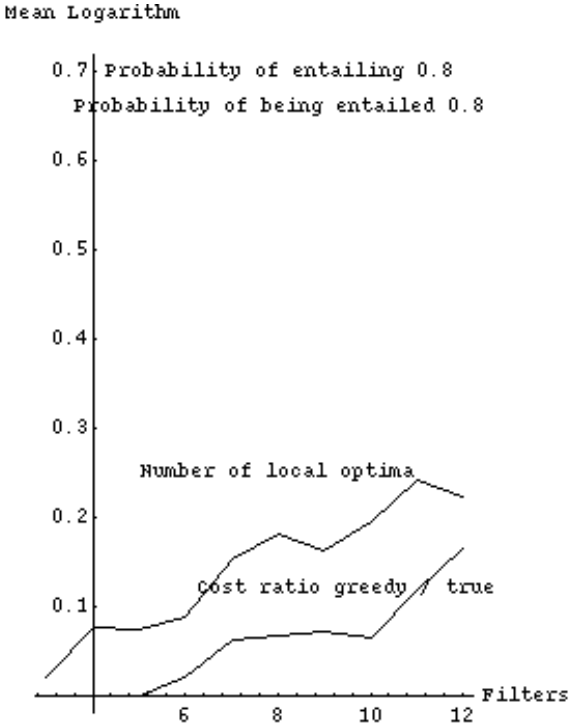

Figure 10: Answer quality: Case 4.

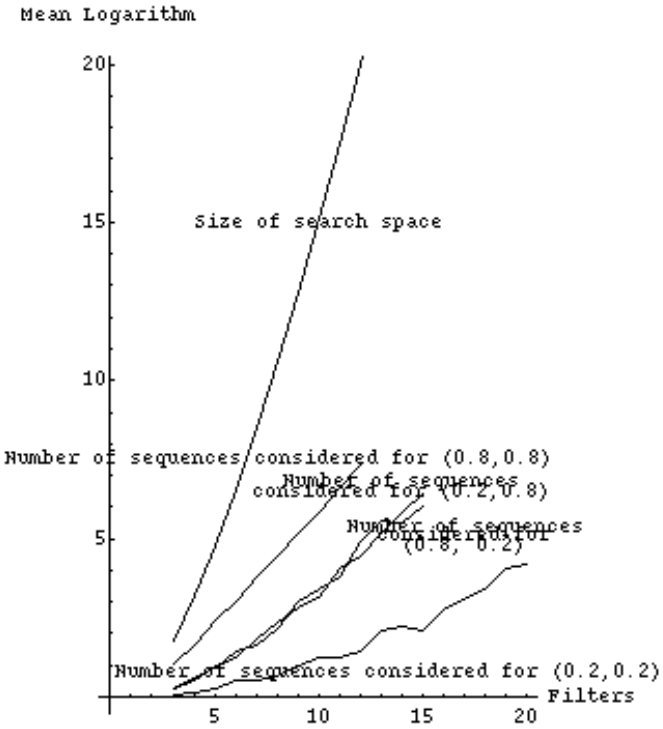

Figure 11: Comparative search difficulty.

Class of logical equivalences	*Optimal independent of costs and probabilities?*	*Guaranteed to work for a greedy (nonbacktracking) algorithm?*
Conjunctive commutivity	*no*	*yes*
Conjunctive elimination of entailed filters	*no*	*no*
Disjunctive commutivity	*no*	*yes*
Disjunctive elimination of entailed filters	*no*	*no*
Redundancy eliminations	*yes*	*yes*
Distributivity factoring	*yes*	*yes*
DeMorgan's Laws inward	*yes*	*no*

Figure 12: Summary of the optimality status of the standard boolean manipulations

---

database universe

C, coarse-grain matcher *P, natural-language parser of query (no joins)* *R, registration data tester*

T, picture-type checker

F, fine-grain semantic

matcher

S, shape analysis

Figure 13: Dependencies between filters in the MARIE-1 system

---

C, coarse-grain matcher (1 processor)

P, natural-language parser (1 processor)

R, registration-data tester (1 processor)

T, picture-type matcher (all processors)

F, fine-grain matcher (all processors)

S, shape analysis (optional, all processors)

Figure 14: Optimal execution plan for the MARIE-1 system, following the analysis in the text

Go up to paper index